IN THE UNITED STATES PATENT & TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS & INTERFERENCES

In re App. No.:	09/449,021	)	PATENT APPLICATION
		)	
Filing Date:	November 24, 1999	)	Art Unit: 2192
		)	
Inventors:	Emmelmann	)	Examiner: C. Kendall
		)	
Title:	*Interactive Server Side Components*	)	
		)	Customer No.: 8665

APPELLANT'S REPLY BRIEF

APPELLANT'S REPLY BRIEF

TABLE OF CONTENT

STATUS OF CLAIMS

Claims 1-2, 22-23, 26, 30, 32-33, 41-42, 59-63, 67-69 and 71-73 were finally rejected in the Office Action dated January 21, 2010, and are the subject of this appeal.

In the Examiner's Answer, claims 6, 8, 51-58, 74-96 and 114-128 were identified as allowable, and claims 3-5, 24-25, 27-29, 31, 43, 64-66 and 70 were objected to but identified as allowable. Although Appellant had therefore requested cancellation of these claims in a separate amendment filed with the original Reply Brief, appellant has also filed a petition concurrently herewith to reinclude parts of these claims in the appeal after being informed during an interview that the filed continuation application in parts might be rejected on the same new grounds as discussed in the new Examiners answer.

Claims 9-21, 34-40, 44-50 and 97-113 have been previously withdrawn as drawn to non-elected subject matter and are formally cancelled in the amendment request. Claim 7 has been previously cancelled.

GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

The grounds of rejection to be reviewed on appeal are:

(1) Whether claims 1, 2, 22, 23, 41, and 42, rejected under 35 U.S.c. 103(a) as being unpatentable over "Responsive Interaction for a Large Web Application", The Meteor Shower Architecture in the WEBWRITER II Editor written by Arturo Crespo et al. (1997), hereinafter WEBWRITER II, in view of Faustini USPN 5,842,020;. and

(2) Whether claims 26, 30, 32, 33, 59 - 63, 67 - 69 and 71 -73 rejected under 35 U.S.c. 103(a) as being unpatentable over "WebWriter: A Browsers -Based Editor for Constructing Web Applications" written by Arturo Crespo et al. (1996), Hereinafter "WEBWRITER" or WEBWRITER I, in view of Faustini USPN 5,842,020.

ARGUMENTS

I. INTRODUCTION

Since new grounds of rejection were raised in the Examiner's Answer dated July 12, 2011, this Reply Brief contains new discussions for most of the claims on appeal. However, the new grounds rejection are still mainly based on the WebWriter prior art, which is discussed in the Appeal Brief and prior Reply Brief dated December 13, 2010,, and appellant still thinks that the examiner significantly misinterprets the WebWriter prior art. Therefore, all the sections of the Appeal Brief and the original Reply Brief that discuss the WebWriter prior art remain relevant, and applicant incorporates the Appeal Brief and the original Reply Brief by reference for those discussions, and refers to specific sections of the Appeal Brief and the original Reply Brief in the arguments below, particularly since the newly cited Faustini patent adds little to the argument against patentability.

The now pending claims on appeal relate to appellant's development of an improved component editor for **editing of server side dynamic web applications** (as opposed to a client side application) **that generate web pages for transmission to and display by a web browser**. Appellant's editor itself is such a server side dynamic web application, running on a server, and operating in a web browser. Appellant's editor provides operations to insert, modify, and delete components denoted on document templates. ***Significantly,*** **appellant's editor runs edited applications and components during editing**. That is, the editor operates on documents that have been generated by a document generator that is running the application being edited. (see the detailed discussion in section A.i, pp. 13-15 of Appeal Brief). Nothing in the portions of the prior art cited in the Answer teach or suggests such a thing. Advantageously, running the application being edited shows the content of dynamically generated parts of the documents during editing. Showing the dynamically generated content helps the developer understand exactly what the user

sees and he can effectively and efficiently make adjustments to document templates to reflect that better understanding.

After the original Reply Brief dated 12/13/2011 and various interviews, the examiner issued a new Answer with new grounds of rejection, citing the Faustini patent in combination with WebWriter I (for some claims) or WebWriter II (for other claims). It thus appears that the Examiner finally agrees with appellant that WebWriter does not **run** the edited application during editing. For example the examiner states "*WebWriter doesn't explicitly disclose that the modifying of documents is to dynamic documents such that the editing is to running applications for dynamic documents.*" (See Answer dated 7/12/2011 at pp. 8, 14 and 26). Thus, the Examiner apparently now cites Faustini for that teaching, and combines it with one or the other of the WebWriter articles. However, appellant disagrees with the Examiner's conclusion that such a combination would make appellant's claims obvious. Further, in appellant's view, the Examiner has failed to make even a *prima facie* case for obviousness of the claims due to the lack of detail regarding the proposed combination in the new grounds of rejection.

The newly cited Faustini patent appears to describe editing of java applets and java components. The java applets and java components have access to the computer display and run while being displayed. Typically, these would be called **client-side components**. However, the portions of Faustini cited in the Answer do not describe generating browser code to communicate with the user, or a document generator that executes components denoted on a document template in order to generate a web page for transmission to and display by a web browser.

Appellant's claims define over Faustini's components by requiring, for example: "*a document generator program running at least part of one of the applications being edited*" (claim 1) or "*a document generator program having instructions ... for executing said components*" (claim 22). The Examiner does not make any effort to describe how these client side components would be incorporated into the WebWriter system. Further, he suggests that Faustini provides a means to run WebWriters' dynamic areas during editing, but does not identify these means. Appellant notes that it is completely different to run WebWriter's components during document generation than to run Faustini's components while the document is being displayed. Appellant

explains in detail why Faustini adds little or nothing to the rejection in Section II.B below.

The WebWriter prior art provides an editor for server side dynamic web applications. It solves the problem of displaying dynamically generated content by providing placeholders during editing, not the dynamically generated content (see the detailed discussion in the Appeal Brief, section A.ii, pp. 15-18, esp. p. 16, 3rd and 4th paragraphs, and p. 17). Appellant discussed the Examiner's misunderstanding of the WebWriter prior art in Section C below.

The Examiner seems to believe that running part of the application being edited by generating browser code on the server during editing is just a missing feature that could easily be added to WebWriter. However, appellant disagrees, and thinks that modifying WebWriter to provide this feature would not be simple, but would require significant changes, possibly to the architecture of the editor, because the editor must avoid interfering with proper execution of the application, because the application needs to be provided with input parameters, and because the editor and application must share the browser without interference. For example, the WebWriter I article describes putting documents displayed for editing into an HTML form, which prevents the documents from containing active forms themselves (which would be needed to provide the edited application with parameters), because nested forms are not allowed in HTML. See section II.C.ii below and section C.ii and C.iii in the appeal brief.

The Examiner may have a fundamental misunderstanding with regard to the term "dynamic content." He now argues that "*Faustini provides the means to edit the dynamic content.*" (Answer at pp. 36, 34). However, appellant believes there are two different kinds of dynamic content - client side dynamic content, which is executed while a document is being displayed, and server side dynamic content, which is executed during document generation. Both WebWriter and appellant are using dynamic content that generates browser code during document generation, while Faustini appears to edit client side dynamic content that runs while the document is being displayed. Because of this fundamental difference, appellant submits that it is not obvious or readily apparent how to use or incorporate Faustini's techniques on server side dynamic content as well.

Appellant believes that a set of recently cancelled claims should also be included in this appeal, and this issue is discussed in Section IV below. This set of claims was cancelled during appeal and re-filed as a continuation upon the Examiner's statement in the original Answer that they were allowable. However, many of the claims have now been rejected by the Examiner in the continuation, and those claims should properly be a part of this appeal.

II. THE CLAIMS ARE PATENTABLE OVER THE CITED COMBINATIONS

A. THE EXAMINER HAS NOT DEMONSTRATED PRIMA FACIE OBVIOUSNESS

The key to supporting any rejection under 35 U.S.C. 103 is the clear articulation of the reasons why the claimed invention would have been obvious. (*KSR International Co. v. Teleflex Inc.*, 550 U.S. 398, 82 USPQ.2d 1385, 1396 (2007); MPEP 2141 *et seq.*).

The Examiner has issued new grounds of rejection, but has failed to provide well articulated reasoning describing a clear correspondence between the elements in the claim and the cited prior art, nor has he described how the prior art would be modified to achieve the proposed combination. Instead, the Examiner draws bare conclusions that are not supported by a detailed review of the cited art, such as "*it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine WEBWRITER and Faustini because it would enable editing each component or property that generates dynamic content as desired and then observe the edited changes live within the environment of WEBWRITER.*" (Answer dated 7/12/11 at p. 9).

The Examiner cites only two small portions of the Faustini patent in describing the combination with WebWriter, namely col. 5: 15-23, which appears to relate to customization of components, and col. 154: 36-48, which appears to describe some part of the user interface. For example, the Examiner repeats the same language each time Faustini is cited, namely: "*Faustini discloses in col. 5:15 – 23, "dynamic editing capability as soon as the component is instantiated or a component editing request is made . . . is also able to edit each component's properties and their limits as desired and then observe the edited changes . . .* "" (Answer dated 7/12/11 at pp. 8-9). However, appellant fails to appreciate how this citation, combined with teachings from one of the WebWriter articles, renders any of appellant's claims obvious. The Examiner appears to assume that the proposed combination would share the advantages of both

WebWriter and Fautsini, without any discussion as to how such a combination could be achieved.

Nevertheless, appellant believes that the issue is clear anyway since some of the claimed features are not taught or suggested in any of the cited portions of WebWriter or Faustini, as discussed in more detail below.

The Examiner also now explicitly ignores the WebWriter page generator, stating that "*only the editor of both versions of WebWriter is used in the rejections. Neither the Page Generator of WebWriter I or that of WebWriter II is used in the rejections.*", (Answer dated 7/12/2011 at p. 32, item 24). This is likely because he now agrees with appellant that the WebWriter Page Generator described in the WebWriter articles is a separate program from the WebWriter Editor running only after editing.

However, since nevertheless some claims require page generator elements in order to run at least part of the application, the examiner must now believe that Faustini provides some part, which provides that function. However, the examiner has failed to discuss this issue at all. According to appellant's understanding, Faustini describes executing a java applet or components that have access to the computer display, but does not describe a document generator that executes components that operate by generating browser code for transmission to and display by a browser program.

The Examiner also cites portions of the WebWriter articles describing the editor, and applying these citations for the claimed document generator, bearing in mind that in appellant's claims, a document generator function is required to execute at least part of the application being edited. On the other hand, the Examiner now seems to agree with appellant that the *WebWriter Editor* and the *WebWriter II Editor* **do not** execute part of the edited application. (Answer at p. 8 ("*Webwriter does not explicitly disclose that the modifying of documents is to dynamic documents such that the editing is to running applications for the dynamic documents.*")(emphasis removed)) The Examiner then repeats the quoted language from column 5 of Faustini, however, as reasoned above, this portion does not seem to teach or suggest running a server based web application that operates by generating browser code and sending that to a web browser.

Thus, appellant believes that the Examiner has not pointed to anything in Faustini that could be used to run a server based application with dynamic areas that

have been created/edited/saved with the *WebWriter Editor*, while the explicit non-citation of the *WebWriter Page Generator* takes away some functional capability for running an edited application. Therefore, the present rejection fails to show the running of the edited, server based web applications, or of components generally, regardless whether the running takes place during editing or after editing.

B. THE FAUSTINI PATENT ADDS LITTLE TO THE DISCUSSION OF PATENTABILITY

Faustini describes a java applet and calls it VJ Tool and notes that "*VJ tool could also be provided as a full application*". (Faustini at col. 10: 13-15 and 20) According to Faustini, "*[t]he user of VJ Tool ... could build or create a Java applet or application from scratch just by using VJ Tool in the manner described herein.*" (*Id.* at col. 10: 21-24). Further, such a tool purports to provide "*dynamic editing of object oriented components used in an object oriented applet or application*" (*Id.* at Abstract, lines 1-2).

As appellant understands it, the VJ Tool works on java applets, components, or applications that have access to the computer display. For example, Faustini states that "[w]*hen a new component is instantiated on either the physical or logical display*", (see Faustini at col 33: 3-4), while appellant's components and WebWriter's dynamic areas have to generate browser code, send it to the browser, and ask the browser to display the generated browser code in order to communicate with the user. Faustini also describes a simple AWT text field, a simple AWT checkbox, etc., and appellant notes that java AWT components seem to operate on client side and have access to the computer display, keyboard and mouse (Faustini at col. 135:10-18 and 33-37).

In appellant's understanding of the cited portions of Faustini, the editing method described cannot be applied to server side components/applications that communicate with the user by sending generated browser code on request to a web browser. For example, Faustini teaches: "*When a new component is instantiated on either the physical or logical display an optional customizer (edit) window is presented if a containerEditor method is defined for the component.*" (Faustini at col. 33:2-5. A browser-based, server-side component/application cannot easily present a window because the component/application does not have direct access to the display, but

instead has to generate browser code and send it to the browser in order to display a window. The cited portion of Faustini seems to be totally unrelated to generating browser code or to a page generator, while the claims require a specific document generator or instructions to generate specific browser code.

Also, Faustini's techniques require a Java-enabled browser in order run java code on the client (see Faustini at col. 8:26), while appellant's editor works with an ordinary browser with HTML and JavaScript.

C. THE EXAMINER CONTINUES TO MISINTERPRET THE WEBWRITER ARTICLES

The WebWriter articles are relevant in that they discuss creating and editing web applications. However, despite much correspondence and discussion regarding these articles, the Examiner continues to read more into these articles than they actually teach or suggest, interpreting them far too broadly in comparison with the claims and disclosure of appellant. The Examiner does seem to finally agree with appellant that neither of the WebWriter articles describes running an application while editing the application.

The WebWriter I article describes a system having an editor, the *WebWriter Editor*, and a document generator, the *WebWriter Page Generator*. The *WebWriter Editor* is a direct manipulation web page editor that is used build and save pages (templates), where the pages may be created with placeholders for dynamic content that are filled-in at runtime. Once the application has been saved (or edited offline), the *WebWriter Page Generator* runs the application and generates pages (documents) for display in response to document requests from the browser.

The WebWriter II article describes an improved editor for the WebWriter system, namely the *WebWriter II Editor*, which uses both server-side and client-side processing to improve operation and efficiency. The *WebWriter II Editor* is a direct manipulation editor, but is still just used to build applications and edit them before running them. The WebWriter II article uses the *WebWriter Page Generator* to "create[s] new pages as a WebWriter-built application runs." (WebWriter II at sec. 1.1).

i. THERE IS NO FUNCTIONAL CONNECTION BETWEEN THE *WEBWRITER PAGE GENERATOR* AND *THE WEBWRITER EDITOR*

As noted above, the WebWriter System has two distinct and separate programs: the *WebWriter Editor* (or *WebWriter II Editor*) and the *WebWriter Page Generator*. The *WebWriter* Editor creates and/or edits web applications, then saves the applications. The *WebWriter Page Generator* runs saved applications. There is no direct functional connection between the two programs. The *WebWriter Editor* does not run the edited application and does not use the *WebWriter Page Generator* to run the edited application. In fact, the *WebWriter Editor* generates pages on its own for the editor without running the application being edited and without the help of the *WebWriter Page Generator*.

This issue has been discussed in depth in the Appeal Brief and the prior Reply Brief. (*See* the original Reply Brief at sec. II.A, particularly sub-sections A.i and A.ii, on pp. 13-18, and the Appeal Brief at sub-sections C.i, C.ii and C.iii, on pp. 40-43).

Although the Examiner states that the new rejection no longer relies on the portion of the WebWriter I article describing the *WebWriter Page Generator* (*See, e.g.,* Answer at p. 32, item 24), it remains unclear how he proposes to run an application being edited. Also, removing the *WebWriter Page Generator* from the rejection suggests that the Examiner believes it is at least a separate, removable part.

The Examiner writes at several places that arguments made in response to prior rejections are moot. (*See, e.g.,* Answer at p. 32, item 24). Appellant disagrees. The section entitled "*The WebWriter Page Generator*" on page 9 of the WebWriter I reference describes running of web applications that have been edited by the *WebWriter Editor*, and running of the edited applications is an important point of appellant's claims. Appellant submits that the Examiner did not drop reliance on the *WebWriter Page Generator* from the rejection because it wasn't needed, but because it was running at the wrong time (at run time, not at editing time) and because it was not connected to the editor as claimed. In addition, the Board is not limited to the Examiner's rejection, and therefore discussions regarding the *WebWriter Page Generator* are still relevant.

ii. THE WEBWRITER EDITOR DOES NOT PERMIT EDITING OF AN APPLICATION RUNNING DURING EDITING

For the *WebWriter Editor,* this issue was a main topic of the Appeal Brief (sections A.i and A.ii) and the original Reply Brief (sections B and E). The Examiner apparently no longer maintains that the *WebWriter Editor* runs the edited application during editing, stating "*WEBWRITER doesn't explicitly disclose that the modifying of documents is to dynamic documents such that the editing is to running applications for the dynamic documents*" and "*WEBWRITER II doesn't explicitly disclose that the editing is to running applications for generated documents.*" (*See* Answer at p. 8 (re: claim 26), p. 14 (re: claim 59), and p. 26 (re: claim 1)). Appellant finds the Examiner's quoted statements above to be unclear and possible incorrect; and further, the Examiner still says in the Answer that the WebWriter Editor works at run-time. (Answer at p. 6 (re: claim 26)).

Appellant refers to sections C.iii and C.ii of the Appeal Brief and section III.A1.c for a discussion of this topic.with regard to WebWriter II. Appellant notes that WebWriter II also seems to use the WebWriter Page Generator for running the application: "*In addition to the Editor, the Webwriter system includes the WebWriter Page Generator, a server based CGI service that creates new pages as a WebWriter – built application runs.*" (See WebWriter II at p. 1507-1508).

Appellant notes that WebWriter actively teaches away from running during editing, by teaching two separate programs, the *WebWrite Editor* and the *WebWriter Page Generator*. The *WebWriter Page Generator* is used at runtime only. The *WebWrite Editor* uses placeholders for dynamic content, which would not be needed otherwise (see appellants arguments to item 26 below). Thus, WebWriter I effectively teaches features that make it possible to edit applications without running them during editing.

Appellant notes that it is not a trivial task to correctly run the application being edited. In fact, appellant submits that it would not be possible using the WebWriter architecture. For example, the *WebWriter Editor* shows a document template as preview not from its usual URL, but instead from an editor-specific URL, and transmits none or the editor's URL parameters instead of the URL parameters required by the

application. Links generated by the application might point the browser somewhere outside the editor and therefore interfere with the correct operation of the editor. In addition to correctly running the application, it also needs to run in a proper execution environment and receive proper input. In contrast, the WebWriter II article teaches a client-side document generator implemented in JavaScript running inside the browser (see WebWriter II at section 4.1.3 and section III.A1.b below), while the WebWriter I article teaches that the dynamic areas are connected to Unix scripts intended for execution on a server computer outside the browser (see WebWriter I at p. 9, section entitled "The Web Writer Page Generator"). Also see discussion of claim 23 below for more details. The *WebWriter Page Generator* also seems to be used with the *WebWriter II Editor* as well. (See WebWriter II at p. 1508).

The Examiner alleges that it would be obvious to run an application being edited by taking WebWriter in combination with Faustini, but does not make clear what is combined or how. For example, with regard to claim 1, which requires "*a document generator program running at least part of one of the applications being edited,*" the examiner writes simply "***The combination of** WebWriter **II and Faustini allows for the editing of JavaScript code that incorporation with the document tree remaps component data into a new desired HTML representation of handles.***" (Answer at pp. 26-27) . The Examiner does not explicitly state that JavaScript code is part of the application or that it is being executed by the document generator or during editing.

Also, the Examiner does not cite to any portions of Faustini that discuss document generation or editing/running of JavaScript code. Thus, it remains unclear exactly how the Examiner is demonstrating that the combination of WebWriter and Faustini shows running part of the application during editing.

In addition, even if the proposed combination shows running of JavaScript code by the document generator described in section 4.1.3 of the WebWriter II article, it would still be irrelevant for claim 1. Claim 1 requires a document generator running at least part of the application outside the browser, while the document generator described in section 4.1.3 of WebWriter II works inside the browser as detailed in section III.A1.b below.

Appellant acknowledges that Faustini describes running and editing components during use of the VJ tool. However, this seems to work only for components that have access to the computer display, e.g., client-side components, and not for components that have to generate browser code to communicate with the user (See section II.B above). As discussed herein, appellant's document generation claims require more than just 'running,' for example, claim 1 recites "*a document generator program running at least part of one of the applications being edited*" and claim 22 recites "*a document generator program having instructions ... for executing said components.*" The portions of Faustini cited by the Examiner appear totally unrelated to document generation. Therefore, applicant submits that Faustini is not effective to suggest running of an application that generates browser code.

The Examiner does briefly discuss execution of the application during editing, noting: "*In response, the Examiner first notes that disclosure of the script placeholder during creation of the document does not mean that the script could not be executed during editing, particularly in view of Faustini as applied in the rejection above.*" (See item 26, p. 33 of the Answer) This is further evidence that the Examiner agrees with appellant that WebWriter alone does not show this feature. And as noted above, it is not trivial to correctly run server-side applications during editing.

iii. THE PREVIEW SHOWN IN WEBWRITER I AND II DOES NOT FUNCTION DURING EDITING AND LOOKS DIFFERENT THAN THE ACTUAL DOCUMENT

This issue has been heavily discussed in the Appeal Brief (see sec. A.ii), and the Examiner seems to admit that the WebWriter articles alone do not teach that by adding "*in the view of Faustini*" (Answer at p.33) and by adding "*Faustini provides the means to edit dynamic content.*" at the end. (Answer at p.34).

The Examiner writes : "T*he examiner takes note that the dynamic content is computed at run time but respectfully asserts that the once the document is created, it would have been obvious to present the entire document with both static and dynamic content in order to show a more realistic view of the document being edited.*" which possibly brings the misunderstanding to a point. (Answer at p. 34. item.26) Appellant thinks that this is the key problem, because run time is after editing as discussed in

previous section ii above, and so the dynamic content is not yet known during editing. So appellant submits that it is not at all obvious to display the actual generated dynamic content during editing, because in WebWriter its known only later. So in applicants understanding the "obvious" solution provided by the examiner would not work and therefore the examiners argument is void.

D. THE APPEAL BRIEF AND PREVIOUS REPLY BRIEF REMAIN RELEVANT

The Examiner replaced his original final rejection, based on a combination of WebWriter I and WebWriter II, with a new rejection, wherein some claims are rejected on a combination WebWriter I and Faustini and others claims are rejected on a combination of WebWriter II and Faustini. On page 32 of the Answer, the Examiner argues that many arguments of the Appeal Brief and previous Reply Brief are now moot because of the changed rejection. The Board, however, is not bound only to the Examiner's rejection, and therefore a thorough discussion of the WebWriter articles can be helpful to a complete understanding of the issues herein.

In addition, for many claims, appellant's reasoning is that neither WebWriter nor Faustini shows a specific feature, and therefore what is discussed in the Appeal Brief and previous Reply Brief regarding the WebWriter articles remains relevant.

It also appears that disagreement remains with regard to what is taught by the WebWriter articles. For example, in the rejection of claim 26, the Examiner does not say that he agrees will appellant that WebWriter does not disclose a similar appearance, but he concentrates only on the issue of running. Also, for claims 2 and 23, the Examiner does not make explicit what claimed elements are not provided by WebWriter. In many arguments, the Examiner discusses WebWriter in detail, then adds a final sentence like: "*Faustini provides the means to do so*", e.g., in the Answer at point 25 on pp. 32-33, at point 26 on p. 24, at point 28 on p. 35, and at point 30 on p. 35. In other places, the Examiner seems to argue without making use of Faustini at all, such as at point 35 on p. 37 and point 27 on p. 34.

For all these reasons, appellant submits that the Appeal Brief and prior Reply Brief remain relevant. Further, although it appears that the Examiner has accepted

many of appellant's arguments, if this should not be the case, or further clarity is required, appellant refers to the Appeal Brief and prior Reply Brief.

III. DETAILED DISCUSSION OF CLAIMS

A. Independent Claims

1. Claim 1

a. Summary

Claim 1 stands rejected as obvious over the combination of WebWriter II and Faustini. However, appellant believes claim 1 is patentable over the cited combination for at least two reasons:

(i) because claim 1 requires a document generator **running outside** the browser, while the Examiner cites in WebWriter II in section 4.1.3 a document generator which seems to run **inside** the browser, and because the cited portions of Faustini for claim 1 do not seem to teach a document generator at all,

(ii) because the claim requires "*a document generator program running at least part of one of the applications being edited and generating the generated documents.*"

The preamble defines *"at least one of the applications generates generated documents for display by the browser program"* and (a) WebWriter II does not provide a document generator running part of the application being edited and (b) the portions of Faustini cited for claim 1 do not seem to teach a document generator or running applications that generate generated documents for display by the browser program.

Neither WebWriter II nor Faustini nor their combination teaches or suggests the claimed combination of a document generator and editor, where the document generator is located external to the browser and provides the functionality to (1) run the

application being edited, and (2) generate documents for display by the browser having editing features.

For a discussion of claim grouping, appellant refers to the Appeal Brief at page 18 and submits that claim 1 should be considered separately.

b. Discussion of argument (i)

Claim 1 requires "*a document generator program ... generating the generated documents"* and the term generated documents is defined in preamble: *"whereupon request by the browser program at least one of the applications generates generated documents for display by the browser program on a display device and responds to the request with the generated documents".* This makes clear that upon a request of the browser, the browser receives **the generated documents** as response, which also makes clear that **the generated documents** are provided from outside to the browser. Since the document generator program is required to generate **the generated documents**, it is clear that it must be running **outside** the browser.

On page 24 of the Answer, the Examiner refers to section 4.1.3 of WebWriter II in his bracketed remarks regarding the document generator element, and so appellant assumes that the Examiner believes that section 4.1.3 discusses the page generation algorithm for the claimed document generator.

In appellant's reading, the document generator presented in section 4.1.3 of WebWriter II is running **inside** the browser, since it is implemented in JavaScript and provided to the browser on an HTML page. (See WebWriter II at p. 1513 "*The preview frame generally contains module preview.html which has three parts (1) Definitions of JavaScript functions to walk the document tree and translate it to html*" and further "*This updates the display without requiring any significant interaction with the server*"). Because WebWriter II shows a document generator running inside the browser, but the claim implicitly requires a document generator running outside the browser, appellant submits that WebWriter II is not effective as prior art for claim 1.

Further, WebWriter II teaches that generated HTML is directly written into the preview frame (see WebWriter II at p. 1513 *"(3) ... which actually causes the new HTML to be written into the preview frame*"), while claim 1 requires that the document generator generate the generated documents and that the application "*responds to the*

request with the generated documents". This also shows that WebWriter II is not effective as prior art for claim 1.

The Examiner also quotes the following from page 1513: "*This updates the display without requiring any significant interaction with the server;*" which might raise the question about the remaining insignificant interaction with the server. The article explains "*because preview.html is cached at the browser*" which in appellant's reading means that only in the relatively rare situations (e.g., the first time) that preview.html is actually downloaded from the server, while normally it is taken from the cache, causing an insignificant server interaction because it happens only rarely. In appellant's reading, the server interaction refers to downloading preview.html, which is, as discussed above, a page containing the document generator itself, and not the output of the document generator as claimed. The article continues: "*and the document tree is converted to HTML as the browser interprets preview.html.*" and so makes clear that the HTML for the document tree is generated inside the browser while the browser executes the javascript code on preview.html. This also makes clear that Webwriter II is not effective as prior art for claim 1.

Appellant further notes that the WebWriter II article describes various other document generator functions producing documents for the other frames of the WebWriter II user interface. For example, the Examiner also cites the document generator in section 4.1.4 of the article, where is says: "*Fig. 7 shows how the process is initiated in the preview frame, requesting the server to send the appropriate HTML file to the* ***object properties frame****" (emphasis added).* The document generator described in section 4.1.4 is irrelevant, however, because it does not address the preview frame. It generates documents showing part of the editor itself and not the generated document the editor operates on, as claimed by appellant with the language "*an editor program dynamically operating on the generated documents displayed.*" In addition, the document generator described in section 4.1.4 seems to be unrelated to the application being edited, and is object-specific: "*This is done through an interface that is object-specific.*" Therefore, the page generator in section 4.1.4 is overcome by the claim language "*at least one of the applications generates generated documents*" and "*a document generator program running at least part of one of the applications being edited and generating the generated documents.*" So, the citation of the document

generator in section 4.1.4 is irrelevant for the claim. Also, see the discussion about point 67 in the Answer at the very end of this document for a clear discussion about what actions WebWriter II performs on the server.

Thus far, appellant has discussed the application of WebWriter II to claim 1 by showing that the page generator described in section 4.1.4 of WebWriter II is running inside the browser while the claim requires a page generator running outside the browser. The Examiner actually rejects the claim on a combination of WebWriter II and Faustini. The Examiner writes on page 26:"***WEBWRITER II*** *doesn't explicitly disclose that the editing is to running applications for generated documents"* and continues "***The combination* of** *WebWriter* ***II and Faustini allows for the editing* of*JavaScript code that incorporation with the document tree remaps component data into* a *new desired HTML representation* of *handles".*** All of this seems to be related to argument (ii) discussed below and does not seem to contribute to the document generator running inside or outside the browser. Also, the cited portion at col. 5: 15 – 23 of Faustini seems to be totally unrelated to document generation. Therefore, appellant submits that claim 1 should be considered patentable over the combination of WebWriter II and Faustini.

c. Discussion of argument (ii)

In appellant's reading, section 4.1.3 of WebWriter II does not teach a document generator that runs at least part of one of the applications being edited, and the Examiner finally seems to agree: "***WEBWRITER II*** *doesn't explicitly disclose that the editing is to running applications for generated documents."* In contrast, the claim language requires "*a document generator program* ***running at least part of one of the applications being edited***" (emphasis added). See also, the appeal brief at sections A.i and A.ii for WebWriter I and section C.iii and C.ii for WebWriter II, and section II.C.ii above.

In addition, as discussed in section (i) above, the claim language requires that the document generator be running outside of the browser. Consequently, the claim language "*a document generator program running at least part of one of the applications being edited"* also must refer to running a part of the application outside the browser. However, as discussed in section (i), the WebWriter II article says "*This*

updates the display without any significant interaction with the server" (see p. 1513), which suggests to appellant that there is actually no execution of the application outside the browser or else it would have been discussed in the article. (See also the discussion regarding point 67 below for a clear discussion about what actions WebWriter II performs on the server.)

The Examiner continues at the end of page 26: "***The combination of*** *WebWriter* ***II and Faustini allows for the editing of JavaScript code that incorporation with the document tree remaps component data into a new desired HTML representation of handles,"*** which frankly, just does not make sense to appellant. The HTML representation of handles appears quite irrelevant to appellant, because the representation of the handles is not important for the claim(s). Possibly, the Examiner thinks about the HTML representation of the component itself and not of the handles. Also, the meaning of "JavaScript code that incorporation with the document tree remaps component data into a new desired HTML representation" is unclear. Maybe the Examiner means that the JavaScript code does the remapping, but as written it says the incorporation does the remapping. Also, it is unclear whether and when the Examiner proposes to execute the scripts.

The cited portions of Faustini seem to neither discuss editing of JavaScript Code, or a document tree, or code that remaps component data into a new desired HTML representation; in fact, the cited portions do not seem to discuss HTML generation at all.

Appellant also notes that scripts in the document tree on the client are irrelevant for the claim. The claim requires: "*a document generator program running at least part of one of the applications being edited and generating the generated documents*" and "*at least one of the applications generates generated documents for display by the browser program*" which makes it clear that the at least part of one of the applications must be executed by a document generator located outside the browser and generating the generated document for display by the browser. WebWriter II, however, has a document generator discussed in section 4.1.3 that runs inside the browser, and the document tree is also inside the browser, as discussed in section 4.1.3 ("*the document tree is a JavaScript data structure built when a page is loaded into the editor and stored in the root window*").

Also, the java applets and components thereof as disclosed by Faustini and discussed in section S.46 and II.B above seem to be irrelevant for the claim, because these seem to be executed inside the browser during display of a document, while the claim requires "*a document generator program running at least part of one of the applications"* outside the browser as discussed in section (i).

d. Traversing remaining part of Examiner reasoning on claim 1

The preamble of claim 1 defines a relevant computing environment to include a client/server data network, wherein a browser program makes a request for documents, and an application program located somewhere else responds to the request by generating and sending the requested documents to the browser.

The Examiner attempts to correlate portions of the WebWriter II article to the preamble (see Answer at p. 23), but the cited portions only discuss the editor itself, not the edited applications, while the preamble of claim 1 provides context with regard to the edited applications.

The Examiner also uses the word "component" several times when discussing WebWriter II and claim 1, for example, saying "*wherein the documents contain components* to *generate content such components are rerun in order* to *display the edited document;* see *the following sections pg.* 1513, *section* 4.1.3." (See Answer at pp. 23-25). Appellant notes that the word "component" does not appear in the reference, neither in section 4.1.1 nor in the later cited section 4.1.3 so it is not clear what exact element of WebWriter II the Examiner refers to when using the word "component."

a) Possibly the Examiner refers to the objects mentioned in section 4.1.3 when discussing components. The article refers to objects that are defined on page 1509: "*We refer to an HTML element in the preview frame as "object""* but also mentions that the templates could contain scripts. In appellant's reading and in disagreement with the Examiners statement from above, there is nothing in the cited portions that talks about objects to generate content or that such objects are (re)run.

b) Possibly the Examiner refers for components to "*JavaScript functions to walk the document tree and translate it into HTML*" as written under heading (1) on page 1513 of the article. If so, then the Examiner's remark is irrelevant for claim 1 since the claim language does not recite components. The claim language recites a part of the

application being edited, however, in appellant's reading, the JavaScript functions to walk the document tree are part of the editor and not part of the application being edited. (See "*The preview frame generally contains module preview.html*" in the beginning of section 4.1.3 on p. 1513, and "*Of the 23 modules composing the WebWriter II editor ...* " on p. 1511, 2nd para.).

Although not used in claim 1, appellant uses the word component quite differently in the context of other claims. These other claims contain claim language that further defines the kind of components claimed, and this is inconsistent with the way the Examiner uses to word component on page 24 of the Answer. This becomes important for the other claims and will be further discussed there.

e. Summary for claim 1

Appellant has given two important reasons why claim 1 should be considered patentable over the cited combination, discussed in section III.A1.b and III.A1.c, including a discussion of the corresponding arguments of the Examiner. Section III.A1.d contains a traversal of the remaining parts of the Examiner's reasoning, which seems to be important for other claims.

2. Claim 22

The Examiner states that the claim limitations are substantially similar in content to claim 1 and are therefore rejected for the same rationale. Appellant disagrees, and notes that claim 22 recites components while claim 1 does not. Other differences include, e.g., editing features or execution of applications in claim 1 versus execution of components in claim 22 or more detailed definitions of applications and the generated documents in claim 1. Appellant also refers to his reasoning for separate patentability in the Appeal Brief on page 26 and notes that this claim should be treated separately. Although the Examiner did not include a separate rejection for claim 22, but refers to his reasoning for claim 1, he nevertheless discusses some aspects of claim 22 in argument section 34 on page 37, which appellant discusses in subsection c further below.

In summary, Appellant thinks that claim 22 is patentable over the cited art because it requires the editor to operate on documents that were generated by the

document generator program, and the document generator program is required to **execute the components**. This is expressed using the claim language "*an editor program operating... **on generated documents**... and a document generator program for executing said components and for generating **the generated documents.***" In addition, the claim language "*having instructions for inserting, deleting, and modifying components on document templates*" makes clear that the components are claimed part of the documents being edited.

Section 4.1.3 of WebWriter II describes the cited document generator, but does not seem to teach execution of components by the document generator as claimed by appellant. Also, the cited portions of Faustini do not seem to teach execution of components by a document generator; in fact, the portions of Faustini cited for claim 1/22 do not seem to mention a document generator at all. It appears to appellant that Faustini's components are executed during document display.

a. WebWriter

In his reasoning for claim 1, the Examiner cited the page generator documented in section 4.1.3 of the WebWriter II reference as prior art for the claimed document generator. In appellant's reading, however, section 4.1.3 of WebWriter II does not teach a document generator that executes components of the applications, and the Examiner finally seems to agree, for example, writing on page 26: "***WEBWRITER II** doesn't explicitly disclose that the editing is to running applications for generated documents.*" This question has also been discussed in the appeal brief (section A.i and A.ii for WebWriter I and C.iii and C.ii for WebWriter II) and the first reply brief mainly with respect to WebWriter I and in section II.C.ii above.

On page 24 the Examiner writes "*(via editing* of *documents in the preview frame, wherein the documents contain components* to *generate content such components are rerun in order* to *display the edited document;* see *the following sections pg.* 1513, *section* 4.1.3". Appellant stresses that the original article neither in section 4.1.1 nor in the later cited section 4.1.3 uses the word "component" so it is not clear what exact element of WebWriter II the Examiner refers to when using the word "component".

a) Possibly the Examiner refers to the objects mentioned in section 4.1.3 when discussing components. The article refers to objects that are defined on page 1509: "*We refer to an HTML element in the preview frame as "object"*" but also mentions that the templates could contain scripts. In appellant's reading and in disagreement with the Examiner's statement from above, there is nothing in the cited portions that talks about objects to generate content or that such objects are (re)run.

b) Possibly the Examiner refers to the JavaScript functions on preview.html described under heading (1) in section 4.1.3. Appellant's believe these functions are part of the editor. Thus, they are not components in sense of claim 22 because the claim requires "*instructions for inserting, deleting, and modifying components on document templates*" while the scripts are a part of the WebWriter II editor and are not being inserted, deleted or modified. Also, preview.html is not a document template or a generated document in sense of claim 22, because it is neither being edited nor being generated but just a fixed part of WebWriter II. (See "*The preview frame generally contains module preview.html*" in the beginning of section 4.1.3 on p. 1513, and "*Of the 23 modules composing the WebWriter II editor ...* " on p. 1511, 2nd para.).

b) Faustini

With regard to the Examiner's statements about the combination of WebWriter II with Faustini, appellant refers to his discussion of claim 1 in section III.A1.c. above.

In contrast to claim 1, however, claim 22 requires components, and Faustini actually seems to discuss some kind of components. As discussed in section I and II.B, Faustini's components are quite different.

The parts of Faustini cited for claim 1/22 also do not seem to describe execution of components by a document generator generating documents understandable by a web browser. In appellant's reading, the components and java applets discussed in Faustini have direct access to the client display, but do not seem to be executed by a document generator generating documents understandable by a web browser.

Because neither WebWriter II nor Faustini teaches execution of the components by the document generator, appellant submits that claim 22 should be considered patentable over the cited combination.

c. Discussion of Examiner's arguments re claim 22

The Examiner writes: *"As per claim 22, arguments relating to execution of at least part of an application during editing have been addressed above."* (Answer at p. 37, point 34). However, claim 22 recites executing components, not executing at least part of an application, which is different. Applicant refers to sections a and b above.

d) Proposed Amendment

With regards to claim 22 appellant refers to a proposed amendment on page 2 of an email from appellants counsel to the Examiner Bullock dated 3/23/2011 which can be found in the interview summary. Appellant thinks that this would make argument (i) as discussed in section III.A1.b for claim 1 applicable for claim 22 as well.

3. Claim 26

Appellant refers to his reasoning for separate patentability in the Appeal Brief on p.28 and notes that this claim should be treated separately.

Claim 26 is directed to an application for modifying dynamic documents. It requires "transforming at least one first document ... into a second document having features which permit editing of the first document." In addition, Claim 26 requires that the second document **appears and functions similar** to the run-time view of the first document.

As discussed in the Appeal Brief (see pp. 16-17, sec. A.ii), dynamic areas in the HTML document are replaced by placeholders during editing. Although these placeholders represent or stand for the dynamic areas, they do not have a similar appearance and are not functional, as shown in the figures of the WebWriter article, reproduced in section A.ii of the Appeal Brief. Claim 26 requires an application for modifying dynamic documents, and in WebWriter, the first document indeed has dynamic areas, but WebWriter shows handles for the dynamic areas which look different than the real content and which are not functional. The Examiner now seems to agree that WebWriter alone does not teach similar appearance nor similar function, and appellant also refers to subsection b below.

Appellant thinks that claim 26 is also patentable over a combination with Faustini, because the portions of Faustini cited for this claim do not teach generating documents for display by a browser from dynamic documents, whereby the term dynamic document is interpreted in the light of appellant's specification, e.g., see page 8, line 22-23. The components and java applets discussed in the portions of Faustini cited in the Examiner's Answer have access to the computer's display and do not generate browser code (see section S.68).

With respect to Faustini the Examiner writes: "*Therefore it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine WEBWRITER and Faustini because it would enable editing* ***each component*** *or property* ***that generates dynamic content*** *as desired and then observe the edited changes live within the environment of WEBWRITER.*" (Answer, pg 8, emphasis added). Appellant disagrees. In appellant's reading, the cited portions of Faustini do not discuss components that generate browser code. Faustini's components seem to have access to the computer display and do not seem to work by generating content as discussed in section II.B. The Examiner repeatedly cites col. 5:15-23 of Faustini, but in appellant's reading, the cited portion appears to be unrelated to document generation or to components that generate dynamic content for display by a browser. It also does not seem to disclose generation of a document that has a similar view or function as another document. Therefore, appellant submits that the Examiner's reasoning is wrong, and the cited portions of Faustini do not seem to disclose what the Examiner suggests. Therefore, the Examiner failed to establish a prima facie case of obviousness and appellant submits that the claim patentable over the cited combination.

a. Traversing the Examiner's rejection step by step

On page 6 of the Answer, the Examiner apparently made a typo, wrongly reproducing claim 26 to omit the word "dynamic." Claim 26 recites in the preamble: "an application to modify **dynamic** documents . . ."

Right off the bat, the Examiner says that the WebWriter Editor allows for editing at runtime. (See Answer at p. 6: "*Web Writer Editor, which runs on the Browser and allows for direct manipulation by a user to displayed web pages* ***at runtime****, e.g.*

modifying dynamic documents" (emphasis added)). Appellant disagrees, and submits that editing takes place not at runtime, but before runtime, as discussed in the appeal brief (see section A.ii) and in WebWriter I (see p. 2: "*the static parts of each page are created in* ***advance*** *by the designer using a text editor*" (emphasis added)).

WebWriter I teaches: "*Once a WebWriter application is created and saved to disk it can be run*" (see p. 6), which makes clear that runtime is after editing. This has been extensively discussed in the appeal brief (see section A.ii) and the reply brief. The Examiner also uses the term "live document" to suggest a document being edited is running (see Answer at p. 6: "left column displays the live document to allow users to edit"), which appellant believes is not used in the WebWriter I reference.

The Examiner further states that " *calling WEBWRITER . . . will invoke WebWriter Editor.*" (Answer at p. 7). Appellant disagrees. WebWriter is a system having two separate programs: the *WebWriter Page Generator* and the *WebWriter Editor*. There is no program named WebWriter itself which could be called or which could invoke the *WebWriter Editor*. (see Abstract of WebWriter I).

b. Similar appearance and similar function in WebWriter I

The issue of similar function and similar appearance with regard to the WebWriter I reference by itself is extensively discussed in the appeal brief (section A.ii, pp. 16-18) and first reply brief. Appellant believes that the Examiner now agrees that WebWriter I does not teach or suggest similar function and similar appearance.

The Examiner does not seem to address similar appearance directly; however, with regard to similar function, the Examiner states: "***WEBWRITER*** *doesn't explicitly disclose that the modifying of documents is to dynamic documents such that the editing is to running applications for the dynamic documents.*" (Answer at p. 8). With regard to similar appearance, the Examiner seems to suggest that "*Faustini . . . provides the means to do so*", without really explaining how or why that would be true. (Answer at p. 33). This does suggest to appellant that the Examiner agrees that WebWriter I alone does not teach similar appearance. However, he confuses the point by saying further: "*With respect to claims 26 and 32, the basis for the rejection is articulated above. Further, since the document presented in the Preview Window for editing is the current form of the document, it appears and functions similar to the first document*

(the document before editing)." (Answer at p. 37). This passage does not mention Faustini, which suggests that maybe the Examiner still thinks that WebWriter I can do it alone? Appellant refers to the extensive discussion of that topic in the appeal brief and reply brief and asserts that these arguments are not moot, but very relevant (also see discussion of point 24).

The Examiner argues that since the document presented in the Preview Window for editing is the current form of the document, it appears and functions similar to the first document. (Answer at p. 37). Appellant disagrees. The document shown in the Preview Window of WebWriter for editing shows only placeholders for dynamic content, not the dynamic content itself, while the runtime view of the first document shows the generated content. Therefore, these two documents would normally not appear similar. Also, in appellant's reading, the document in the Preview Window of WebWriter is not functional. This has all been discussed in great detail in the appeal brief (section A.ii) and the first reply brief, and appellant thought that the Examiner had agreed to and accepted this difference, as stated in II.C.iii.

Appellant notes that the Examiner discusses this issue in points 25, 26 and 28 in the Answer as well, and appellant discusses all this in section II.C.iii above.

4. Claim 59

Appellant refers to the Appeal Brief including the reasoning for the separate patentability of claim 59 and submits that the claim therefore should be considered separately.

Claim 59 is directed to a software development system for dynamic web documents having an editor and a document generator program. The Examiner seems to recite the WebWriter I editor as prior art for both the claimed editor and for claimed the document generator.

The claim requires "*a document generator program having instructions for generating generated documents from dynamic web documents which look and function similar to the end user's view of the documents with the addition of editing features*" and the Examiner cites the WebWriter I editor as prior art. It has already been extensively discussed that the documents displayed by the WebWriter I editor do not look and function similar to the end user's view, because WebWriter I displays

placeholders for dynamic areas. (See section A.ii of the appeal brief and section II.C.III above). Appellant also refers to section III.A3.b for a discussion of the same issue on claim 26.

The Examiner seems to at least partly agree by stating "***WEBWRITER*** *doesn't explicitly disclose that the modifying of documents is to dynamic documents such that the editing is to running applications for the dynamic documents wherein the dynamic components once edited are dynamically viewed.*" (Answer at p. 14) However, the Examiner then cites Faustini and concludes "it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine WEBWRITER and Faustini because it would enable editing each component or property that generates dynamic content as desired and then observe the edited changes live within the environment of WEBWRITER." (*Id.* at p. 15, citing Faustini at col. 5:15-23).

Appellant disagrees. Appellant can find nothing in the cited portion that would enable the WebWriter I editor to generate pages that look and function similar to the end user view, as is required by the claim language. Further, the cited portions of Faustini do not seem to discuss components that generate dynamic content. Faustini's components appear to have access to the computer display and not to work by generating browser code, as discussed in section II.B. The cited portion appears to be unrelated to document generation or to components that generate dynamic content, nor does it seem to disclose generation of a document that has a specific view or function. Therefore, appellant submits that the Examiner's reasoning is wrong and the cited portions of Faustini for claim 59 do not seem to disclose what the Examiner suggests. Therefore, the Examiner failed to establish a prima facie case of obviousness and appellant submits the claim is patentable over the cited combination.

a. Traversing the Examiners Rejection Step by Step

The Examiner writes: "***calling WEBWRITER . . . will invoke*** *WebWriter* ***Editor.***" (Answer at p. 11) Appellant disagrees. The WebWriter system has two programs: the *WebWriter Page Generator* and the *WebWriter Editor*. There is no program named WebWriter itself which could be called or which could invoke the *WebWriter Editor*. The portions of WebWriter cited on page 11 seem to discuss editing, but do not seem to cover the language of claim 59: "*said web documents operating by*

being transformed into an end user's view upon a request by a web browser," which appellant believes is mainly described on page 9 of the WebWriter article in the section entitled "*WebWriter Page Generator.*" However, the Examiner now states that his rejection no longer depends on the WebWriter Page Generator. (Answer at point 24 on p. 32).

The Examiner also uses the term "live document" to suggest a document being edited is running (see Answer at p. 11: "left column displays the live document to allow users to edit"), which appellant believes is not used in the WebWriter I reference.

b. First instructions requesting the document generator program

Claim 59 requires that the editor program having first instructions for requesting that the document generator program processes a dynamic web document. Appellant does not see such instructions in WebWriter I, instead, the editor program just seems to generate HTML documents with links and forms so that the document generator is requested by the browser when the user interacts with these links or form buttons. Appellant notes that WebWriter II uses the JavaScript reload function and therefore has instructions to issue a request, but only for downloading preview.html and not for processing the dynamic web document. Also, the cited portions of Faustini do not seem to disclose instructions for requesting a document generator, and therefore appellant submits that the claim is patentable over the cited combination.

c. Discussion of Alternative Rejections

As a possible and more logical looking alternative, the Examiner could have cited the *WebWriter Page Generator* as prior art for the claimed document generator. This has been discussed in the appeal brief discussion of claim 59 as an alternative (although the previous rejection was to cite the *WebWriter Page Generator* as prior art for the claimed page generator and the claimed editor).

However, appellant believes that the Examiner is right in not citing the *WebWriter Page Generator* since it runs *after* editing, as explained in sections A.i, A.ii and C.i of the appeal brief, and so it is clear that the *WebWriter Page Generator* does not receive requests from the WebWriter Editor during editing. In contrast, claim 59 requires *"the editor*

program comprising first instructions for requesting the document generator to process a dynamic web document leading to a generated document."

Combining the *WebWriter Editor* or the *WebWriter Page Generator* with elements from Faustini does not, in appellant's understanding, change the basic idea taught by the WebWriter article that the WebWriter editor generates the pages it needs during editing on its own, with its own page generation mechanism.

B. The Dependent Claims

1. Claim 2

a) Traversing the examiners Reasoning

Claim 2 is dependent on claim 1, and for all the same reasons is patentable over the cited combination. In addition, claim 2 requires a plurality of components with various limitations, including "*wherein the document generator executes selected components on document templates.*" The examiner seems to mention various elements in the prior art as 'components' but none of them seems to be executed by the document generator as required by the claim. In the following, appellant discusses the examiner's reasoning step by step.

On page 27 of the Answer, the examiner cites portions of WebWriter II, section 4.1.1. Unfortunately, the quoted material is missing closing quotes so that it is not clear without comparing to the original article that the examiner added the following text which is not part of the WebWriter II article: "*(e.g. into a template document for display)* ***via editing* of *documents in the preview frame, wherein the documents contain components* to *generate content such components are rerun in order* to *display the edited document;* see *the following sections pg. 1513, section* 4.1.3,".**

The examiner included the same statement in the discussion of claim 1, and therefore appellant has already discussed this in section III.A1.d. It was also discussed with regard to claim 22, but in a different way, because claim 22 recites components (just as claim 2 does), and therefore appellant mainly refers to that discussion in section III.A2.a above.

The examiner recites Faustini without actually saying what he finds to be missing in WebWriter II. For claim 1, the Examiner stated "**WEBWRITER II** doesn't explicitly disclose that the editing is to running applications for generated documents" while he left out that phrase for claim 2. However, the examiner included the same arguments as for claim 1, and therefore appellant refers to sections III.A1.b and III.A1.c for a discussion of these arguments. Appellant also refers to the discussion for claim 22 in section III.A2.b, which was rejected with the same arguments and which has a similar limitation requiring a document generator for executing the components. The

components mentioned in the cited portions of Faustini do not seem to be executed by a document generator as required by claim 2.

In summary appellant thinks that the examiner did show any prior art for the claim limitation "*the document generator executes selected components on document templates"* and submits that the claim should therefore be considered patentable over the cited combination.

b) Additional Reasons

Reading the limitations of base claim 1 together with claim 2 limitations further narrows the claim: claim 2 requires "*wherein the document generator executes selected components on document templates*" and base claim 1 requires the document generator generating the generated documents, and the preamble says "*generates generated documents for display by the browser program and responds to the request with the generated documents*." This makes clear that the document generator is required to run outside the browser (see III.A1.b) and to provide documents to the browser. Because the document generator is required to execute selected components, it should be clear that the components are required to execute outside the browser as well. The examiner does not seem to address this distinction (although appellant thinks it was discussed in an interview), instead he refers to Javascript code on the HTML pages *"The HTML page sent to frame could be static HTML . . . or an HTML page that includes JavaScript code"* which makes it clear that the article refers the JavaScript code for execution inside the web browser. Appellant also notes that Faustinis java applets seem to run inside the browser and for that reason are not as relevant to the claim.

In summary, appellant thinks that in addition to the reasons for patentability given in section a), claim 1 and claim 2 limitations read together implicitly require a document generator executing selected components outside the browser. This makes it even more convincing in applicant's eyes because many things described in WebWriter II and Faustini work inside the browser.

2. Claim 23

The examiner merely copied reasoning for claim 2. Appellant therefore does the same. Claim 23 depends on Claim 22 and for all the same reasons is patentable over the

cited combination. Claim 23 contains a new limitation "*wherein the editor program operates a functional application in an edit mode permitting editing of said application directly in the web browser*" and for that reason, appellant submits that the claim needs to be considered separately.

Appellant notes that the reasoning provided on pages 29-31 of the Answer discusses scripts, components and running part of an application, but not operating a functional application, as required by claim 23. For example, on page 31 of the Answer, the examiner's conclusion that *"The combination of WebWriter II and Faustini allows for the editing of JavaScript code that incorporation with the document tree remaps component data into a new desired HTML representation of handles"* (which appellant disagrees with as discussed above) is also not sufficient to teach the claimed recitation: "*the editor program operates a functional application in an edit mode permitting editing of said application directly in the web browser.*" The examiner does discuss this limitation as item number 44 on page 39, but refers to previous reasoning for editing an application while the application is being run, and only discusses operation of a functional application in an edit mode during editing.

The examiner further writes: "*since the document being displayed in the Preview Window of the WebWriter-Faustini combination is the current form of the document as interpreted HTML, the displayed document is functional.*" The examiner apparently refers to WebWriter II at page 1509, which states: "*In editing mode the WebWriter II Editor displays the current page as interpreted HTML . . . .*" In applicant's reading, this means the current page is displayed in some altered form, i.e., as interpreted HTML. The exact meaning of "interpreted HTML" is not clear, however, appellant notes that the this portion of the article says HTML and not HTML with scripts, while for other documents it does explicitly mention scripts, e.g., in the last paragraph of page 1511. In any case, appellant disagrees that the cited portions show that the current page as interpreted HTML contains scripts of the current page (if it has some) and so is functional in that respect. Appellant also refers to section 4.1.3 of the article as describing the page generator generating the document for display in the preview frame. "The preview frame generally contains module preview.html which has ... JavaScript functions to walk the document tree and translate it into HTML ... which actually causes the new HTML to be written into the preview frame. "

The examiner seems to suggest that simply showing the current page unchanged as HTML inside the editor would render it functional. However, appellant submits that inside the editor the page is displayed in a different environment, e.g., inside a frame and from another URL, which might render relative URLs in links or in scripts non-functional.

Appellant also likes to point to HTML links and form actions. It is the purpose of both to make the browser display a page in a selected frame or the current frame. WebWriter II, however, requires that the preview frame show 'preview.html' as discussed in the beginning of section 4.1.3. Therefore, appellant thinks that clicking on a link or form in WebWriter II would interfere with editing, and therefore it makes sense that the links and forms are non-functional during editing.

Also, it appears that WebWriter II edits only one page at a time. "Load" and "Save" buttons are described on page 1512, and upon using the load button the document tree is created. In contrast, a functional web application can have multiple pages and skip pages via links or from buttons.

Appellant also refers to his reasoning in the appeal brief on p.49-50.

The examiner concludes: "The obviousness of including dynamic content or applications as part of the document in this window has been addressed above." Appellant has argued elsewhere that it is not obvious, and therefore the displayed document cannot be functional because of the missing dynamically generated parts.

In summary, appellant thinks the examiner failed to show prior art for "*the editor program operates a functional application in an edit mode permitting editing of said application directly in the web browser,*" while appellant gave various reasons why the pages shown by WebWriter II do not appear functional. Appellant therefore submits that claim 23 is patentable over the cited combination.

3. Claim 30

Claims 27-40 depend from Claim 26, and for all the same reasons, these claims are patentable over the cited combination.

With respect to claim 30, appellant refers to his reasoning in the Appeal Brief, which is still valid. In addition, the examiner now refers to fig 2 of the reference and *Cut*

/ Copy / Insert functionality that inherently function as scripts to insert copy or remove objects from the web page. There is nothing describing where those scripts are located, i.e., on the server or inside the generated document. However, reading base claim 26 together with claim 30 requires scripts in the second document, i.e., the document that is generated (see Appeal Brief page 54 for details):

"... transforming at least one first document retrieved from the document store into a second document having **features** ..." (claim 26) and "**the features** include scripts" (claim 30).

The Examiner did not include reasoning about which documents contain the scripts and therefore failed to construct a *prima facie* case of obviousness.

In applicant's reading, the generated documents do not seem to contain scripts. On page 8 of the WebWriter I article, the WebWriter Editor section discloses just HTML code. Also, on page 1, the WebWriter Editor is described as "*a CGI program that runs in any browser that supports tables,*" which seems to indicate that it might run in a browser not supporting scripts as well. For all these reasons, appellant submits that claim 30 should be considered patentable over the cited WebWriter I and Faustini prior art.

Appellant nevertheless notes that WebWriter II has scripts, while the claim was rejected on WebWriter I.

4. Claims 41-42

Appellant thinks claim 41-42 should be patentable over the cited combination for the same reasons as their base claim 1.

With respect to point 42 on page 38 of the Answer, appellant thinks that claim 1 requires the document generator to be outside the browser by specifying that it generates the generated documents, noting that the preamble includes "*responds to the request with the generated documents.*" The main argument in the appeal brief nevertheless focuses on claim 41 requiring a client part on the client computer and arguing that WebWriter I does not have such a client part. Since the claim is now rejected on WebWriter II, this might no longer be relevant since WebWriter II has a client part. However, in applicant's reading, the document generator described in section 4.1.3 of WebWriter II is operating inside the browser and therefore the editor is

not operating on **the** generated documents because of the preamble saying "*responds to the request with **the** generated documents*".

The examiner also says: "*In fact the only limitation requiring that a particular program be part of a particular location within the network is that the browser program be run on the client computer. Other programs are just required to be part of the network. Neither is such a limitation is found in claims 41-42*" (Answer at p. 39). Appellant notes that claim 41 says "*client part for execution on the **client computer***" (emphasis added) and claim 42 says "*client part comprises instructions for execution during editing that are automatically downloaded from the **server computer***" (emphasis added).

5. General discussion on claims dependent on claim 59

The discussion of the following dependent claims heavily depend on what element of the prior art is held against the document generator recited in claim 59 and referenced in several dependent claims.

In the previous rejection, the examiner applied the *WebWriter Page Generator* against the claimed document generator and the claimed editor, while in the present rejection, he appears to use the *WebWriter Editor* instead (see discussion of claim 59 in section III.A4) as prior art for the claimed editor and the claimed page generator. For sake of completeness, appellant discusses in section III.A4.c the missing case, wherein the *WebWriter Editor* is used against the claimed editor and the *WebWriter Page Generator* is used against the claimed page generator.

For dependent claim 60, the examiner again takes the *WebWriter Page Generator* as prior art for the claimed page generator, while the situation for the other depend claims is not as clear. Appellant will therefore discuss in the following sections both cases for most of the dependent claims.

Claims 60-73 depend from Claim 59, and for all the same reasons, these claims are patentable over the cited combination.

6. Claim 60

The previous rejection of claim 60 was based on a combination of WebWriter I and WebWriter II, but has now been replaced by a combination of WebWriter I and Faustini. However, the examiner did not cite Faustini directly for Claim 60, just for base claim 59. Therefore, appellant provides some new reasoning, but refers to the appeal brief for additional reasoning as why the claim should be considered separately.

The examiner writes "***WEBWRITER in the Abstract*** *shows the WebWriter Editor program operating in the Browser and a Web generator CGI program on the server, thereby it is obvious that the user interface* of *the browser which shows the editor allows for part* of *the editor program to run on the client computer."* Appellant disagrees and thinks that WebWriter I only sends HTML pages without scripts to the client computer (as discussed with claim 30), and thus in applicant's interpretation, there is no part of the editor running on the client computer. In contrast, the claim requires the editor program to be at least partly running on the client computer.

The examiner's reasoning refers the *WebWriter Page Generator*, although arguments page 32, item 24, say that the current rejection no longer relies on it. Appellant thinks this is inconsistent with the discussion of claim 59, where the examiner uses the *WebWriter Editor* against the claimed document generator.

Claims 61-63 depend from Claim 60, and for all the same reasons, these claims are patentable over the cited combination.

7. Claim 61

Appellant refers to the Appeal Brief including the reasoning for the separate patentability of claim 61 and submits that this claim should be considered separately.

Appellant notes that the reasoning provided for claim 61 in the Appeal Brief handles both cases (a) wherein the *WebWriter Page Generator* is held against the claimed page generator as in the previous rejection, and (b) wherein the *WebWriter Editor* is held against the claimed page generator as in the current rejection.

Claim 61 requires "instructions for execution during the document generation to collect edit information" whereby claim 59 requires the document generation to generate generated documents which look and function similar to the end user's view.

As discussed in the appeal brief, the WebWriter page generator does not seem to teach instructions to collect edit-information. (See WebWriter at p. 9). On the other hand, the *WebWriter Editor* does not have a document generation function that generates generated documents which look and function similar to the end user's view, as discussed with regard to claim 59 in section III.A4.

For all these reasons appellant submits that the claim should be considered patentable over the cited combination of WebWriter I and Faustini.

8. Claim 63

Appellant refers to the Appeal Brief including the reasoning for the separate patentability of claim 63.

As discussed in section III.A4.b, the *WebWriter Editor* does not seem to have instructions for performing a request, it just seems to generate links and form buttons that, when pressed, cause the browser to perform a request.

Claim 63 requires "*instructions for automatically repeating the request that the document generator processes the dynamic web document when required.*" Appellant does not see any such instructions described in WebWriter I. The present rejection recites large portions of the WebWriter I prior art without pointing to any specific instructions. (Answer at p. 18). Also, the cited part of Faustini does not point to any specific instructions. Therefore, appellant thinks the examiner failed to establish a prima facie case of obviousness and submits that the claim should be considered patentable over the cited combination of WebWriter I and Faustini.

9. Claim 67

Appellant refers to the Appeal Brief including the reasoning for the separate patentability of claim 67 and submits that this claim should be considered separately. Appellant refers to and incorporates the reasoning provided for claim 67 in the Appeal Brief.

With regard to the characterization of the document as interpreted HTML, appellant refers to the discussion of claim 23 in section III.A2. However, since claim 67 is rejected on WebWriter I (and not WebWriter II), the case is even more clear as discussed in section A.ii of the appeal brief, and as can be seen on the figures from the

WebWriter I article that shows that the view does not look similar. Since the claim requires that the view looks similar to the end-user view except for editing features, appellant submits that the claim should be considered patentable over the cited combination of WebWriter I and Faustini.

10. Claim 68

Appellant refers to the Appeal Brief including the reasoning for the separate patentability of claim 68 and submits that this claim should be considered separately. Appellant submits that the reasoning for claim 61 applies here as well because both claims recite collecting editing information, but notes that claim 68 is dependent on claim 59 while claim 61 is dependent on clam 60.

11. Claim 69

Claim 69 depends from Claim 68, and for all the same reasons, claim 69 is patentable over the cited combination.

12. Claim 72

The examiner now provides a general citation of various sections of the WebWriter I article and a portion of Faustini. In applicant's reading, the cited portions do not discuss "*instructions for initiating a reload in the browser*" as recited in claim 72. Applicant's understanding is that "reload" is a specific function of a web browser. Appellant believes that the phrase "reload in the browser" as used in claim 72 cannot be interpreted any different than the standard reload function supplied by a web browser, especially since reload is used in the description that same way. However, the examiner did not give a specific citation of where WebWriter I or Faustini uses a reload, and therefore he has not made a prima facie case of obviousness. Thus, the claim should be considered patentable over the cited combination of WebWriter I and Faustini.

Appellant notes that WebWriter II seems to use a "reload" as discussed in the last paragraph of section 4.1.3. However, because the examiner chose to use WebWriter I and Faustini for this claim set, this fact is irrelevant for the current rejection. Still, appellant notes that WebWriter II uses reload to load the "preview.html" page, while

claim 72 requires "*the first instructions comprise seventh instructions for initiating a reload in the browser*" and claim 59 "*first instructions for requesting that the document generator program processes a dynamic web document during editing thereby resulting in a generated document.*"

12. Claim 73

In the Appeal Brief, appellant argued that claim 73 should be considered separately because it requires the important distinctive feature of displaying without requesting that the document generator generates a document.

In applicant's understanding, WebWriter I displays all information on a single web page, the current document in the left column and the user interface in the right column. (See p. 8, 2nd paragraph, section "The WebWriter Editor"). Thus, the display of any new information, e.g., on at least one element, requires the editor to regenerate a new document with the new information. The examiner argues "*It is noted that if no changes are made to a previously regenerated document and the information is solely displayed, it is inherent to the system that no document is regenerated since only changes allow for the document to be regenerated.*" Appellant thinks that the examiners statement (1) almost never applies, simply because the display of new information means that the document generated by the editor changes, and (2) is not correct, at least the examiner did not provide any citation to support his statement, and appellant believes that the articles teaches that the editor regenerates a page regardless whether the same or a different document was generated before. (WebWriter at p. 8).

A note in the WebWriter II article describes disadvantages of the WebWriter I editor: "*Clicking on a handle simultaneously selects it and deselects the old selection. In the original WebWriter Editor this required* ***a server round-trip*** *to redraw the page appropriately.*" (WebWriter II at p. 154). In contrast, claim 73 requires "*to display information on at least one element ... without requesting that the document generator program generates a document.*"

Finally, the examiner argues: "*In addition, all changes automatically initiate a redisplay such that the user does not request anything* of *the document generator program.*" Appellant notes that the claim language does not require the user to request anything and thinks that claim 73 applies to automatically initiated requests.

In summary, appellant submits that WebWriter I needs to do the requesting in order to display information as claimed, while in contrast, the claim requires "without requesting" and therefore appellant submits that the claim should be considered patentable over the cited combination of WebWriter I and Faustini.

IV. THE EXAMINER'S REBUTTAL TO ARGUMENTS IS NOT CONVINCING

With regard to point 25. of the examiners answer:

The examiner states: "*that it would have been obvious to the person of ordinary skill in the art to display dynamic areas of the document in its actual appearance during editing sessions instead of just the static components in order to provide the user with a realistic version of the document being edited. Faustini as discussed in the body of the rejection provides the means to do so.*" This statement does not express any disagreement with applicant's discussion of the WebWriter prior art in the appeal brief. The examiner has now created the obviousness combination with Faustini and refers to the body of the rejection. However, appellant does not think that this topic was sufficiently developed in the rejection as discussed above. In particular, the examiner did not identify or cite the specific "means" of Faustini he refers to.

As discussed in section II.B, appellant thinks that the portions of Faustini cited in the examiners answer fail to show (i) any means to run a server based web application that generates browser code to display a user interface or (ii) any means to generate browser code in order to display the dynamic areas in their actual appearance. As discussed in section II.C.III, appellant thinks that it is actually a problem to display the dynamic areas in their actual appearance, and that WebWriter avoids that problem by teaching the use of placeholders instead.

With regard to point 26. of the examiners answer:

The examiner states various things he considers "would have been obvious" and concludes that Faustini provides the means to edit dynamic content. Appellant

understand this not as a disagreement with appellants characterization of the WebWriter prior art in section A.ii. Instead, this section seems to discuss items the examiner considers as obvious in combination with Faustini.

Appellant indeed argues that the WebWriter articles fail to disclose execution of the application being edited during editing. The examiner then writes that "*...placeholder during creation of the document does not mean that the script could not be executed during editing, particularly in view of Faustini . . . .*". (1) The examiner seems to express that WebWriter could be extended with that functionality, but the Examiner does not contradict applicant's statement that WebWriter as is does not have the functionality. (2) In applicant's understanding, placeholders are used to indicate where the actual content is missing. If the actual content was intended to be shown, there would be no need to show placeholders. So the fact that WebWriter teaches placeholders seems to indicate that the authors of Webwriter were aware of the problem and decided to put some effort into showing placeholders. Thus, appellant thinks they are teaching away from showing the actual content by providing the placeholders as an alternative.solution to showing actual content.

Appellant disagrees that Faustini provides that feature, as previously discussed at item number 25 and section II.B and II.C.ii.

The examiner then includes arguments leading to the conclusion that "*it would have been obvious to include dynamic content for editing as well as static content.*" Specifically, he argues "*that the dynamic content is computed at run-time but respectfully asserts that the once the document is created, it would have been obvious to present the entire document with both static and dynamic content.*" Appellant thinks that this nicely exposes the problem that, at least in WebWriter, run-time is after editing and therefore the evaluated dynamic content is only known at runtime, i.e., after editing, and so it is not obvious to show it during editing because it is created later. So, the obvious solutions the examiner seems to have in mind would simply not work.

The examiner finally states "*Faustini provides the means to edit dynamic content.*" Appellant notes (1) editing dynamic content and showing dynamic content during editing are different, and possibly the examiner meant to discuss the latter? (2) the examiner did not specifically identify what elements of Faustini he refers to as the

"means" and did not give any citations for them, (3) the cited portions of Faustini do not seem to disclose editing of dynamic content in the same way that term is used by Appellant and by WebWriter. The cited portions of Faustini seem to edit java applets or components, which are programs that are downloaded to the client and executed there, directly displaying output on the display. The examiner fails to explain how Faustini could handle dynamic areas as introduced by WebWriter. For further discussion of Faustini, appellant refers to item number 25 and section II.B, II.C.ii and II.C.iii.

With regard to point 27. of the examiners answer:

The examiner writes "with respect to arguments relating to Placing Dynamic Areas on page 16 of the Brief the Examiner notes that this section relates creation of document to be presented to the user, see page 2 The WebWriter application model."

However, this confused appellant since page 16 of the appeal brief does not cite to page 2 of WebWriter but to pages 4 and 6. The arguments presented in the appeal brief on page 16 apply to the creation of new template pages as well as to editing existing template pages.

At the end of his statement, the examiner writes "*and notes that the means to do so exists.*" Appellant assumes that the "means" refers back to Faustini, and that the examiner is again saying that things become obvious by combination with Faustini. Appellant refers to section II.B and II.A.

With regard to point 28. of the examiners answer:

The Examiner notes that the sections cited by appellant on page 16-17 of the appeal brief relate to creation of a document and not to editing of an existing document. Appellant disagrees and does not understand how the examiner comes to that conclusion. These arguments apply to the creation of new template pages as well as to editing existing template pages.

The examiner discusses the case of creation of a new document and of editing of an existing document separately. In the first case, the examiner agrees that WebWriter includes a place holder. In the second case, the examiner comes to the conclusion that it would have been obvious to present any dynamic content. However, it is not clear what

the examiner actually means by dynamic content, the script/application or its actual output.

Possibly the examiner is referring to page 8 of WebWriter I , where it says: "*The user creates an output area by ... WebWriter then adds a placeholder for the output area at the current insertion point. The output area object will be replaced at runtime by the output of a script.*" In applicant's reading, the last phrase makes clear that the output of a script will be shown at run-time, and further: "*Once a WebWriter application is created and saved to disk it can be run*" (see p. 6) makes clear that run-time is after editing. (See also section A.ii of the appeal brief). Appellant thinks that WebWriter shows placeholders regardless of whether it is editing a new or an existing document or dynamic object.

The examiner writes "*It would have been obvious to present any dynamic content currently included in the document and to allow the user to edit that content as well."* Appellant notes that the WebWriter editor edits template pages. The template pages might have dynamic areas which e.g. contain scripts. Dynamic areas are filled by the output of the script at runtime resulting in new temporary pages that are send to the browser. So in that sense the document that is being edited, i.e. the template page, never contains the actual output of the script. If the examiner refers by dynamic content to the script or application itself (as the brackes in point 26 seem to indicate) appellant stresses that beneficially his editor shows actually the output of the scripts during editing. If the examiner by dynamic content refers to the actual output of the script the examiners statement talking about "*dynamic content currently included in the document*" does not make any sense. The actual generated dynamic content is never in the template page, just dynamic areas with e.g. scripts , but not the actual generated dynamic content itself. In contrast appellants editor shows the actual generated dynamic content during editing, although it is editing document templates.

With regard to point 29 of the examiners answer:

Appellant refers to section II.D.

With regard to point 30 of the examiners answer:

With respect to arguments about the document generator being moot, appellant refers to sections II.C.i.and II.D.

The examiner writes: "*In addition, Appellant argues that the prior art fails to disclose use of editing features in pages generated from at least partly running edited application. The Examiner must respectfully disagree.*" Then the examiner argues the combination of Faustini and WebWriter. Thus, appellant assumes that the examiner agrees with appellant's characterization of the WebWriter prior art, but not in combination with Faustini. The examiner writes: "*Faustini provides the means to edit the dynamic content.*" However, the Examiner did not identify what means he is referring to and also did not give specific citations in Faustini describing those means. The citations given just seem to discuss features or user interface of Faustini, but no specific means, as discussed in section II.B above. Appellant does not think combining with Faustini would provide running of WebWriter's dynamic areas. Faustini works with components that have access to the computer display as discussed in I.

Appellant has argued that the *WebWriter Page Generator* does not generate documents having editing features. The *WebWriter Editor*, on the other hand, does not run the application being edited. According to the new rejection, the *WebWriter Page Generator* is no longer used, so apparently the Examiner tries to take the "running" from Faustini. However, Faustini does not seem to disclose the form of running required by the claims, as discussed above.

The examiner then writes: "*If the document being edited includes dynamic content or applications, it would have been obvious to run this content while the document is being displayed in order to provide a more realistic view to the user.*" As discussed already, appellant disagrees. (1) The examiner failed to create a prima facie case of obviousness by not clearly specifying and citing the means in Faustini to combine with WebWriter and by not clearly describing the combination. (2) The problem is not obvious because "*the dynamic content included in the template and computed at run time*" is not yet known during editing and therefore cannot easily be displayed to the user during editing. In is unclear how the examiner the proposed combination could solve this problem. Also, in applicant's reading of the cited portions, Faustini works on client side components and java applets that have access to the screen and no need to generate browser code. Thus, Faustini does not teach running dynamic

content during document generation. In contrast, claim 1 requires "*a document generator program running at least part of one of the applications.*"

With regard to point 31. of the examiners answer:

Appellant refers to section II.D.

With regard to point 32. of the examiners answer:

Appellant is not sure what the examiner cites as prior art for the claimed components. In his rejection of claim 1 on page 24, the examiner uses the word "component" several times, while the reference uses the word "object." (WebWriter II at p. 1509: "We refer to an HTML element in the preview frame as "object"). Therefore, appellant's arguments in section B.ii of the appeal brief with regard to HTML buttons as components are still relevant.

In section B.ii of the appeal brief, appellant writes: "Applicant therefore assumes that the examiner interprets scripts specified for the dynamic areas as prior art for the components recited in applicant's claims," which has not been addressed by the examiner. Appellant would like to know what element(s) of the cited prior art the examiner holds up in correspondence for the claimed components, in order the check if that element fulfils all the requirements of the claims.

With regard to point 33 of the examiners answer:

The Examiner says: "For these reasons the Examiner asserts that claim 1 is not patentable over the prior art, but it is unclear to appellant what the examiner refers to as "these reasons."

With regard to point 34 of the examiners answer:

This is discussed together with the claim 22 rejection.

With regard to point 35 of the examiners answer:

This is discussed together with the claim 26 rejection.

With regard to point 36 of the examiners answer:

The examiner comments in response to section C.vi of the appeal brief and the limitation: "Components may cooperate with the Editor.". Appellant notes that this is different from editing of application or components and has not yet been discussed.

With regard to point 42 of the examiners answer:

This is discussed together with the claim 41 rejection.

With regard to point 44 of the examiners answer:

This is discussed together with the claim 23 rejection.

With regard to point 48 of the examiners answer:

The examiner states with respect to claim 33 that "the user is capable of editing the preview page and via the browser, sending changes remotely such that the new updated webpage is regenerated with the incorporated changes" without giving any supporting citations. Appellant notes that claim 33 is rejected based on WebWriter I and Faustini, and disagrees with the examiner's characterization above. In applicant's reading, there is no separate preview page in WebWriter I, but the current page is displayed as interpreted HTML in the left column (see Fig 2.). In the right column, the editor shows HTML fields and buttons that the user can fill out and press. Then, the corresponding HTML form data is sent to the server. The editor is running on the server performing the requested editing operations. (see left column on page 9 in section The WebWriter Editor).

The examiner writes: "*Therefore, the combination allows for a user to change components that generate dynamic content and for such components to rerun remotely and regenerate a new web page in the preview frame for the user to view.*" It is totally unclear how the examiner comes to such a conclusion and the examiner did not cite any portion showing components that (re)run remotely and regenerate a new web page. In applicant's reading, running of WebWriter's dynamic areas is performed by the *WebWriter Page Generator*, which is not explicitly cited by the examiner (see point 24 in the answer). The cited portions of Faustini seem to show components that have access to the display and therefore have no need to generate a web page. In addition,

appellant notes that claim 33 is rejected based on WebWriter I while point 48 argues about features of WebWriter II, e.g., the preview frame.

With regard to point 50 of the examiners answer:

The examiner writes "*Therefore, the combination allows for a user to change components, as needed, that generate dynamic content and for such components to be automatically rerun remotely and regenerate a new web page based on the requested changes in the preview frame for the user to view.*" Appellant submits that it would be impossible to draw such a conclusion since the examiner wasn't reasoning about components before. In addition, the examiner did not give any citations in support of this statement. In applicant's reading, running of WebWriters dynamic areas is performed by the *WebWriter Page Generator*, which is expressly not cited in the current rejection, and the cited portions of Faustini seem to show components that have access to the display and therefore have no need to generate a web page.

Appellant also notes that, with regard to claim 63, WebWriter I does not have a preview frame, which is a feature of WebWriter II.

With regard to point 52 of the examiners answer:

Appellant notes that claim 61 read together with base claim 59 requires a "*a document generator program having instructions for generating generated documents from dynamic web documents which look and function similar to the end user's view of the documents with the addition of editing features"* and claim 61 in addition requires "*fourth instructions for execution during the document generation to collect edit-information for use by the editor program.*" As discussed in the previous reply brief and apparently accepted by the examiner, the *WebWriter Editor* alone is not a page generator that generates "*web documents which look and function similar to the end user's view of the documents"* as claimed (see section II.C.III). Thus, appellant would expect citations to Faustini from the Examiner in order to describe and support the missing functionality.

With regard to point 53 of the examiners answer:

This is discussed together with the claim 60 rejection.

With regard to point 54 of the examiners answer:

Appellant refers to section II.C.III and A.ii of the appeal brief.

With regard to point 56 of the examiners answer:

In claim 73, the requesting refers to the editor or the eighth instructions. The examiner somehow seems to believe that the claim refers to a user requesting something. The term "without" as used in the claim refers to page 27, lines 19-21 of the specification.

With regard to point 59. of the examiners answer:

Please refer to section II.C.i and II.D for a discussion on moot arguments about the document generator.

With regard to point 60. of the examiners answer:

Appellant had argued that it is the function of the *WebWriter Page Generator* to run the edited application. Since some claims require running during editing, appellant asked the examiner to show that this running takes place during editing. In the new Answer at point 60, the examiner states that previous arguments with respect to the *WebWriter Page Generator* are moot, because the *WebWriter Page Generator* is no longer being used in the rejection. Appellant thinks that this is illogical. Since the claims require running of the edited application, the examiner needs to cite some part in the prior art that accomplishes this task. The examiner has not. Although he purports to rely on Faustini, he gives only a few citations referring to the features of Faustini, but nothing describing a document generator. Appellant also refers to section II.A.

With regard to point 61. of the examiners answer:

The examiner refers to the current rejection and arguments, appellant thinks the rejection does not properly address the issues and arguments regarding document generation.

With regard to point 62. of the examiners answer:

Appellant notes that the running function of the *WebWriter Page Generator* is still required by some of the claims and that the current rejection does not recite any element in the prior art that performs this function. Also, see the discussion of point 60 and section II.A.

With regard to point 63. of the examiners answer:

For a discussion on the *WebWriter Page Generator* function, appellant refers to point 60, point 62 and section II.A.

With regard to point 64. of the examiners answer:

The examiner points to sections already discussed.

With regard to point 65. of the examiners answer:

Appellant disagrees with the examiners characterization, since reload is a specific, well known and standardized function of a web browser, and thus the term cannot be interpreted in some other way.

With regard to point 67. of the examiners answer:

In section C.iii of the appeal brief, appellant argued that WebWriter II performs most of the editing operations (except for loading the document before editing and saving the document after editing) locally on the client. The examiner disagrees in point 67. Appellant also cites section 4.1 of WebWriter II at page 1511, 3rd paragraph: "*The CGI modules provide server-side services such as loading files, parsing HTML, saving files and setting up the environment for the HTML modules at start up. The Javascript modules handle displaying the edited page in the preview frame, selecting the current object and copying and pasting of objects.*" Also on page 1513: "*After the user makes an edit to the document the screen is redisplayed by the calling the Javascript reload() method in the preview frame. This updates the display without requiring any significant interaction with the server (because preview.html is cached at the browser and the document tree is converted to HTML as the browser interprets preview.html).*"

After re-reading the last citations, appellant is still convinced that in order to perform an edit operation except load (including parsing) and save (i.e. to perform an edit as citation says) a server interaction is rarely needed (without significant server interaction), e.g., in those rare cases in that the browser did not yet cache certain parts.

The examiner gives a citation from section 4.1.2 that discusses some server interaction, however, section 4.1.2 is entitled "Loading and saving pages: using the server to access files" and therefore appellant thinks the cited remarks about server operations mostly include loading and saving.

The examiner also mentions the display mode global variable, however section 4.1.2 makes clear that modification of display mode executes locally: "*For example the Hide Handles Button executes locally*" and redisplaying of the preview frame takes place mostly locally as well as previously discussed. Thus appellant does not see where the examiner sees any server interaction.

The examiner also cites global structures in section 4.1.1, which in applicant's reading are used on the client side for various javascript modules to communicate with each other "*pages with JavaScript can collaborate with one another via global data structures and functions placed in the top level page of the browser*". They are also initialized during initialization and loading as discussed in section 4.1.1 and 4.1.2, but appellant does not see what additional server interaction the examiner refers to by mentioning the global structures.

Appellant's assumption that most of the editing operations except load and save are performed with few server interaction does not mean that WebWriter II is not a distributed program. Appellant is still convinced that WebWriter II is optimized to perform the main editing process, wherein the editor performs in a loop user requested editing operations with no or very few server interactions.

In contrast, applicant's editor performs modifications to the document being edited on the server and (re)executes that application document on the server in order to redisplay it during editing. That is not possible with the WebWriter II structure, because the document tree appears to be stored on the client (section 4.1.1) and seems to be transferred to the server only at the save operation (section 4.1.2). And because the changed document is transferred to the server only at the save operation, it can only be executed on the server afterwards.

To summarize point 67, the examiner gave various arguments disagreeing with applicants statement that WebWriter II performs most of the editing operations on the client computer (except for loading the document before editing and saving the document after editing). Appellant, however, sticks to his statement (noting that parsing HTML is part of the loading) since the summary section of WebWriter II on page 1516 almost identically supports applicant's statement: "*The server downloads to the browser a parsed version of the HTML page to be edited; the browser executes JavaScript code to handle editing operations on that page ... The server is only used for fetching and saving files, parsing HTML, and starting up the system.*"

Respectfully submitted,

Date: 12 September 2011 By: /Richard A. Nebb/
Richard A. Nebb
Reg. No. 33,540

DERGOSITS & NOAH LLP
3 Embarcadero Center, Suite 410
San Francisco, California 94111
Telephone: 415.705.6377
Facsimile: 415. 705.6383
Email: rnebb@dergnoah.com